scVAE

Release 2.1.4

Christopher Heje Grønbech, Maximillian Fornitz Vording

Jun 30, 2020

CONTENTS

scVAE is a command-line tool for modelling single-cell transcript counts using variational auto-encoders.

scVAE was developed by Christopher Heje Grønbech and Maximillian Fornitz Vording with Christopher continuing its development. The methods used are described and examined in *Grønbech et al. (2020)*.

CHAPTER

ONE

CONTENTS

1.1 User Guide

scVAE model count data, primarily single-cell gene transcript counts, using variational auto-encoders (*Kingma and Welling, 2014*; *Rezende et al., 2014*).

1.1.1 Installing scVAE

scVAE requires Python 3.6–3.7, which can be installed in several ways, for example, using Miniconda.

With Python in place, scVAE can be installed using pip:

```
$ python3 -m pip install scvae
```

1.1.2 Using scVAE

In general, scVAE is used in the following way:

```
$ scvae $COMMAND $DATA_SET
```

where `$COMMAND` can be `analyse` (data analysis), `train` (model training), or `evaluate` (model evaluation and analysis). `$DATA_SET` is a path to a data set file or a short name for a data set.

By default, data are placed and cached in the subfolder `data/`, models are saved in the subfolder `models/`, and analyses are saved in the subfolder `analyses/`.

In the following, the most relevant options are described. Use the help option to list all options for each command:

```
$ scvae $COMMAND --help
```

1.1.2.1 Data sets

Several data sets are already included in scVAE:

- `Macosko-MRC`: GSE63472.
- **`10x-MBC`: 1.3 Million Brain Cells from E18 Mice from 10x Genomics.**
  - `10x-MBC-20k`: 20 000 sampled cells.
- `10x-PBMC-PP`: Nine data sets of purified PBMC populations from 10x Genomics as specified in *Grønbech et al. (2020)*.

- `10x-PBMC-68k`: Fresh 68k PBMCs (Donor A) from 10x Genomics.
- `TCGA-RSEM`: "transcript expression RNAseq - TOIL RSEM expected_count" data set from the TCGA Pan-Cancer (PANCAN) cohort.

Data sets will be cached in the data directory, which defaults to `data/`. This can be changed using the option `--data-directory` (or `-D`).

Be aware that it might take some time to load and preprocess the data the first time for large data sets. Also note that to load and analyse the `10x-MBC` data set, 47 GB of memory is required (32 GB for the original data set in sparse representation and 15 GB for the reconstructed test set in dense representation).

The default model can be trained on, for example, the `10x-PBMC-PP` data set like this:

```
$ scvae train 10x-PBMC-PP
```

Custom data sets

scVAE can read Loom files, and it can read a dense data matrix from a TSV file or a sparse one from a HDF5 file without further configuration. As an example, a data set in Loom format can be imported and modelled in the following way:

```
$ scvae train data_set.loom
```

The TSV files can be compressed using gzip, but each row should represent a cell or sample and each column a gene (for the reverse case, see below). If a header row and/or a header column are provided, they are used as gene IDs/names and/or cell/sample names, respectively.

For Loom files, scVAE follows Loompy's conventions: each column represent a cell or sample and each row a gene. Cell or sample names are specified using the column attribute `CellID` (or just `Cell`), and the row attribute `Gene` is used for gene names.

HDF5 files should include a single directory containing arrays for the sparse matrix (with names as for SciPy's CSR/CSC sparse matrix format: `data`, `indices`, `indptr`, `shape`). Arrays for example/cell and feature/gene names are also supported with a varieity of naming conventions: for example, `barcodes`, `cells`, `samples`, `examples` for the former; `genes` and `features` for the latter. If either name array or both are present, scVAE will try to orient the matrix to match their dimensions.

scVAE also supports the following formats (supplied using the `--format` option):

- `10x`: Output format for 10x Genomics's Cell Ranger.
- `gtex`: Format for data sets from GTEx.
- `matrix_ebf`: (gzip compressed) TSV file with cells/samples/examples as rows and gene/features as columns (examples-by-features).
- `matrix_fbe`: (gzip compressed) TSV file with gene/features as rows and cells/samples/examples as columns (features-by-examples).

The last of these formats can be used to read a TSV file, which is in reverse order of the default case:

```
$ scvae train data_set.tsv.gz --format matrix_fbe
```

Using the Loom format, included cell types and batch indices can also be imported without further configuration by using the column attributes "ClusterName" and "BatchID", respectively.

Cell types for other formats can be imported in TSV format by instead providing a JSON file with a `values` field with the filename for the read counts, a `labels` field with the filename the cell types, and `format` field with the format.

A JSON file for a GTEx data set would look like this:

```
{
   "values": "GTEx_Analysis_2016-01-15_v7_RNASeQCv1.1.8_gene_reads.gct.gz",
   "labels": "GTEx_v7_Annotations_SampleAttributesDS.txt",
   "format": "gtex"
}
```

Naming this file `gtex.json`, the GTEx data set can then be imported and modelled:

```
$ scvae train gtex.json
```

Withheld data

Any data set can be split into a training, a validation, and a test set using the `--split-data-set` option:

```
$ scvae train $DATA_SET --split-data-set
```

Then, the training set is used to train the model, the validation set is used for early stopping as well as finding the best model parameters, and the test set is used when evaluating the model.

The data set can be split either randomly (`random`) or in the sequence in which it already is[1] (`sequential`). This is done by specifying either value using the option `--splitting-method`:

```
$ scvae train $DATA_SET --split-data-set --splitting-method random
```

The fraction of the data set used for the training and validation sets is set using the option `--splitting-fraction`:

```
$ scvae train $DATA_SET --split-data-set --splitting-fraction 0.9
```

This option also determines the fraction of the training and validation sets used when training a model. The above command will then split the data sets into training, validation, and test sets using a 81 %- 9 %-10 % split.

1.1.2.2 Training a model

The command `train` is used to train a model on a data set:

```
$ scvae train $DATA_SET
```

By default, a VAE model with a Poisson likelihood function, two-dimensional latent variable, and one hidden layer of 100 units will be trained on the specified data set for 200 epochs with a learning rate of 10^{-4}.

The default model can be changed by using the following options:

- `-m`: The model type, either `VAE` or `GMVAE`.
- **`-r`: Likelihood function (or reconstruction distribution):**
  - `poisson`,
  - `negative_binomial`,
  - `zero_inflated_poisson`,
  - `zero_inflated_negative_binomial`,

[1] With the first part becoming the training set, the second part the validation set, and the remaining part the test set.

  - `constrained_poisson`,
  - `bernoulli`,
  - `gaussian`, and
  - `log_normal`.

- `-k`: The threshold for modelling low counts using discrete probabilities and high counts using a shifted likelihood function (denoted by $k_{\max}$ in *Grønbech et al., 2020*). This turns the likelihood function into a corresponding piecewise categorical likehood function.
- `-q`: The latent prior distribution. For the VAE model, this can only be a normal isotropic Gaussian distribution (`gaussian`) or one with unit variance (`unit_variance_gaussian`). For the GMVAE model, this can either be a Gaussian-mixture model with a diagonal covariance matrix (`gaussian_mixture`) or a full covariance matrix (`full_covariance_gaussian_mixture`). Note that a full covariance matrix should only be used for simpler GMVAE models.
- `--prior-probabilites-method`: Method for how to set the mixture coefficients for the latent prior distribution of the GMVAE model. They can be fixed to either uniform values (`uniform`) or inferred values from labelled data (`infer`), or they can be learnt by the model (`learn`).
- `-l`: The dimension of the latent variable.
- `-H`: The number of hidden units in each layer separated by spaces. For example, `-H 200 100` will make both the inference (encoder) and the generative (decoder) networks two-layered with the first inference layer and the last generative layer consisting of 200 hidden units and the last inference layer and the first generative layer consisting of 100 hidden units.
- `-K`: The number of components for the GMVAE (if possible, this is inferred from labelled data, but it can be overridden using this option).
- `-w`: The number of epochs during the start of training with a linear weight on the KL divergence (the warm-up optimisation scheme described in *Grønbech et al., 2020*). This weight is gradually increased linearly from 0 to 1 for this number of epochs.
- `--batch-correction`: Perform batch correction if batch indices are available in data set (currently only possible with Loom data sets).

The training procedure can be changed using the following options (only applicable to the `train` command):

- `-e`: The number of epochs to train the model.
- `--learning-rate`: The learning rate of the model. The model is optimised using the Adam optimisation algorithm (*Kingma and Ba, 2015*).

A GMVAE model with a negative binomial likelihood function, a 100-dimensional latent variable, two hidden layers of each 100 units, and 200 epochs using the warm-up scheme is trained for 500 epochs on the `10x-PBMC-PP` data set like this:

```
$ scvae train 10x-PBMC-PP -m GMVAE -l 100 -H 100 100 -w 200 -e 500
```

Trained models are saved to the subdirectory `models/` by default. This can be changed using the option `--models-directory` (or `-M`).

1.1.2.3 Evaluating a model

The command `evaluate` is used to evaluate a model on a data set:

```
$ scvae evalaute $DATA_SET
```

Note the model has to have been trained already on the same data set.

The model is specified in the same way as when training the model, and the model will be evaluated at the last epoch to which it was trained. If withheld data were used, the model will also be evaluated at the early-stopping epoch and epoch with the most optimal marginal log-likelihood lower bound (if available). A number of analyses are conducted of the models and results, and these saved in the subdirectory `analyses/`. This can be changed using the option `--analyses-directory` (or `-A`). If you want the tool to perform all available analyses, you can use this option and argument: `--included-analyses all`.

Cells can be clustered and cell types can be predicted using the option `--prediction-method`. Currently only *k*-means clustering (`kmeans`) is supported. The GMVAE clusters cells and predict cell types using its built-in density-based clustering by default.

To visualise the data sets or latent spaces thereof, these are decomposed using a decomposition method. By default, this method is PCA. This can be changed using the option `--decomposition-methods`, and as the name implies, multiple methods can be specified: PCA (`pca`), ICA (`ica`), SVD (`svd`), and *t*-SNE (`tsne`).

Decompositions of the data sets and of the latent values as well as predictions and the latent values themselves are also saved to compressed TSV files in the same directory.

The GMVAE model trained in the previous section is evaluated with PCA and *t*-SNE decomposition methods like this:

```
$ scvae evaluate 10x-PBMC-PP -m GMVAE -l 100 -H 100 100 -w 200 --decomposition-
↪methods pca tsne
```

1.1.2.4 Examples

To reproduce the main results from *Grønbech et al. (2020)*, you can run the following commands:

- Combined PBMC data set from 10x Genomics:

```
$ scvae train 10x-PBMC-PP --split-data-set -m GMVAE -r negative_binomial -l 100 -
↪H 100 100 -w 200 -e 500
$ scvae evaluate 10x-PBMC-PP --split-data-set -m GMVAE -r negative_binomial -l␣
↪100 -H 100 100 -w 200 --decomposition-methods pca tsne
```

- TCGA data set:

```
$ scvae train TCGA-RSEM --map-features --feature-selection keep_highest_variances␣
↪5000 --split-data-set -m GMVAE -r negative_binomial -l 50 -H 1000 1000 -e 500
$ scvae evaluate TCGA-RSEM --map-features --feature-selection keep_highest_
↪variances 5000 --split-data-set -m GMVAE -r negative_binomial -l 50 -H 1000␣
↪1000 --decomposition-methods pca tsne
```

1.1.3 Tutorial

Say you have a data set consisting of:

- single-cell transcript counts a file called `"transcript_counts.tsv.gz"` with genes as rows and cells as columns, and
- associated cell types in file called `"cell_types.tsv"`.

To load these, you make a JSON file with the following contents:

```
{
   "values": "transcript_counts.tsv.gz",
   "labels": "cell_types.tsv",
   "format": "matrix_fbe"
}
```

(See *Custom data sets* for more loading options.)

You then save the JSON file in the same directory as the TSV files with a memorable name like `"data_set.json"`.

To load and split this data set with scVAE and train a GMVAE model with a Poisson distribution on the training set, you run the following command in the same directory:

```
$ scvae train data_set.json --split-data-set -m GMVAE -r poisson
```

(See *Training a model* for more model options.)

You evaluate this model on the test set using the following command:

```
$ scvae evaluate data_set.json --split-data-set -m GMVAE -r poisson
```

The resulting plots are saved in a subfolder called `"analyses"`. If you want *t*-SNE plots, you use this command instead:

```
$ scvae evaluate data_set.json --split-data-set -m GMVAE -r poisson --decomposition-
↪methods tsne
```

1.2 License

1.3 Programming Interface

1.3.1 Data module

class scvae.data.**DataSet**(*input_file_or_name*, *data_format=None*, *title=None*, *specifications=None*, *values=None*, *labels=None*, *example_names=None*, *feature_names=None*, *batch_indices=None*, *feature_selection=None*, *example_filter=None*, *preprocessing_methods=None*, *directory=None*, ***kwargs*)

Data set class for working with scVAE.

To easily handle values, labels, metadata, and so on for data sets, scVAE uses this class. Other data formats will have to be converted to it.

Parameters

- **input_file_or_name** (*str*) – Path to a data set file or a title for a supported data set (see *Data sets*).
- **data_format** (*str, optional*) – Format used to store data set (see *Custom data sets*).
- **title** (*str, optional*) – Title of data set for use in, e.g., plots.
- **specifications** (*dict, optional*) – Metadata for data set.
- **values** (*2-d NumPy array, optional*) – Matrix for (count) values with rows representing examples/cells and columns features/genes.
- **labels** (*1-d NumPy array, optional*) – List of labels for examples/cells in the same order as for `values`.
- **example_names** (*1-d NumPy array, optional*) – List of names for examples/cells in the same order as for `values`.
- **feature_names** (*1-d NumPy array, optional*) – List of names for features/genes in the same order as for `values`.
- **batch_indices** (*1-d NumPy array, optional*) – List of batch indices for examples/cells in the same order as for `values`.
- **feature_selection** (*list, optional*) – Method and parameters for feature selection in a list.
- **example_filter** (*list, optional*) – Method and parameters for example filtering in a list.
- **preprocessing_methods** (*list, optional*) – Ordered list of preprocessing methods applied to (count) values: `"normalise"` (each feature/gene), `"log"`, and `"exp"`.
- **directory** (*str, optional*) – Directory where data set is saved.

name
Short name for data set used in filenames.

title
Title of data set for use in, e.g., plots.

specifications
Metadata for data set. If a JSON file was provided, this would contain the contents.

data_format
Format used to store data set.

terms
Dictionary of terms to use for, e.g., `"example"` (cell), `"feature"` (gene), and `"class"` (cell type).

values
Matrix for (count) values with rows representing examples/cells and columns features/genes.

labels
List of labels for examples/cells in the same order as for *values*.

example_names
List of names for examples/cells in the same order as for *values*.

feature_names
List of names for features/genes in the same order as for *values*.

batch_indices
List of batch indices for examples/cells in the same order as for *values*.

number_of_examples
The number of examples/cells.

number_of_features
The number of features/genes.

number_of_classes
The number of classes/cell types.

feature_selection_method
The method used for selecting features.

feature_selection_parameters
List of parameters for the feature selection method.

example_filter_method
The method used for filtering examples.

example_filter_parameters
List of parameters for the example filtering method.

kind
The kind of data set: `"full"`, `"training"`, `"validation"`, or `"test"`.

version
The version of the data set: `"original"`, `"reconstructed"`, or latent (`"z"` or `"y"`).

property number_of_values
Total number of (count) values in matrix.

load()
Load data set.

split(*method=None*, *fraction=None*)
Split data set into subsets.

The data set is split into a training set to train a model, a validation set to validate the model during training, and a test set to evaluate the model after training.

Parameters

- **method** (*str, optional*) – The method to use: `"random"` or `"sequential"`.

- **fraction** (*float, optional*) – The fraction to use for training and, optionally, validation.

Returns Training, validation, and test sets.

clear()
Clear data set.

1.3.2 Models module

class scvae.models.**VariationalAutoencoder**(*feature_size*, *latent_size=None*, *hidden_sizes=None*, *reconstruction_distribution=None*, *number_of_reconstruction_classes=None*, *latent_distribution=None*, *minibatch_normalisation=None*, *batch_correction=None*, *number_of_batches=None*, *number_of_warm_up_epochs=None*, *log_directory=None*, ***kwargs*)

Variational auto-encoder class.

Parameters

- **feature_size** (*int*) – The number of features/genes in the data set to model.
- **latent_size** (*int*) – The number of dimensions to use for the latent space.
- **hidden_sizes** (*list(int)*) – A list of the number of units in each hidden layer of both the inference (encoder) and the generative (decoder) networks. The number of layers in each network is thus the length of this list. For the inference network, the order of the hidden layers is the same as for the list, while for the generative network, it is the reverse.
- **reconstruction_distribution** (*str, optional*) – The name of the reconstruction distribution (or likelihood function; see *Training a model*)
- **number_of_reconstruction_classes** (*int, optional*) – The number of counts to model directly, starting from zero (see *Training a model*).
- **latent_distribution** (*str, optional*) – The name of the latent prior distribution: `"gaussian"` or `"unit_variance_gaussian"` (see *Training a model*).
- **minibatch_normalisation** (*bool, optional*) – If `True`, normalise each random minibatch of data when training or evaluating the model.
- **batch_correction** (*bool, optional*) – If `True`, and if batches are present in data set to model, perform batch correction.
- **number_of_batches** (*int, optional*) – The number of batches in the data set to model. Required, if `batch_correction` is `True`.
- **number_of_warm_up_epochs** (*int, optional*) – The number of epochs during the start of training with a linear weight on the KL divergence. This weight is gradually increased linearly from 0 to 1 for this number of epochs.
- **log_directory** (*str, optional*) – Directory where model is saved.

feature_size
The number of features/genes which can be modelled.

latent_size
The number of dimensions of the latent space.

hidden_sizes
A list of the number of units in each hidden layer of both the inference (encoder) and the generative (decoder) networks. The number of layers in each network is thus the length of this list. For the inference network, the order of the hidden layers is the same as for the list, while for the generative network, it is the reverse.

reconstruction_distribution
An instance of the reconstruction distribution (or likelihood function) class used by the model.

number_of_reconstruction_classes
The number of counts modelled directly, starting from zero.

latent_distribution
An instance of the latent prior distribution class used by the model.

minibatch_normalisation
If `True`, normalise each random minibatch of data when training or evaluating the model.

batch_correction
If `True`, and if batches are present in data set to model, perform batch correction.

number_of_batches
The number of batches in the data set to model, when `batch_correction` is `True`.

number_of_warm_up_epochs
The number of epochs during the start of training with a linear weight on the KL divergence. This weight is gradually increased linearly from 0 to 1 for this number of epochs.

property name
Short name for model used in filenames.

property description
Description of model.

property parameters
Trainable parameters in the model.

train(*training_set*, *validation_set=None*, *number_of_epochs=None*, *minibatch_size=None*, *learning_rate=None*, *run_id=None*, *new_run=None*, *reset_training=None*, ***kwargs*)
Train model.

Parameters

- **training_set** (*DataSet*) – Data set used to train model.
- **validation_set** (*DataSet, optional*) – Data set used to validate model during training, if given.
- **number_of_epochs** (*int, optional*) – The number of epochs to train the model.
- **minibatch_size** (*int, optional*) – The size of the random minibatches used at each step of training.
- **learning_rate** (*float, optional*) – The learning rate used at each step of training.
- **run_id** (*str, optional*) – ID used to identify a certain run of the model.
- **new_run** (*bool, optional*) – If `True`, train a model anew as a separate run with an automatically generated ID.

- **reset_training** (*bool, optional*) – If `True`, reset model by removing saved parameters for the model.

sample(*sample_size=None*, *minibatch_size=None*, *run_id=None*, *use_early_stopping_model=False*, *use_best_model=False*)
Sample from trained model.

Parameters

- **sample_size** (*int, optional*) – The number of samples to draw from the model.
- **minibatch_size** (*int, optional*) – The size of the random minibatches used at each step of training.
- **run_id** (*str, optional*) – ID used to identify a certain run of the model.
- **use_early_stopping_model** (*bool, optional*) – If `True`, use model parameters, when early stopping triggered during training. Defaults to `False`.
- **use_best_model** (*bool, optional*) – If `True`, use model parameters, which resulted in the best performance on validation set during training. Defaults to `False`.

Returns A data set of generated examples/cells as well as a dictionary of data sets of samples for the two latent variables.

evaluate(*evaluation_set*, *minibatch_size=None*, *run_id=None*, *use_early_stopping_model=False*, *use_best_model=False*, ***kwargs*)
Evaluate trained model

Parameters

- **evaluation_set** (`DataSet`) – Data set used to evaluate model.
- **minibatch_size** (*int, optional*) – The size of the random minibatches used at each step of training.
- **run_id** (*str, optional*) – ID used to identify a certain run of the model.
- **use_early_stopping_model** (*bool, optional*) – If `True`, use model parameters, when early stopping triggered during training. Defaults to `False`.
- **use_best_model** (*bool, optional*) – If `True`, use model parameters, which resulted in the best performance on validation set during training. Defaults to `False`.

Returns A data set of reconstructed examples/cells as well as a data set of the latent variable (wrapped in a dictionary for compatibility with *GaussianMixtureVariationalAutoencoder*).

class scvae.models.**GaussianMixtureVariationalAutoencoder**(*feature_size*, *latent_size=None*, *hidden_sizes=None*, *reconstruction_distribution=None*, *number_of_reconstruction_classes=None*, *latent_distribution=None*, *prior_probabilities_method=None*, *prior_probabilities=None*, *number_of_latent_clusters=None*, *minibatch_normalisation=None*, *batch_correction=None*, *number_of_batches=None*, *number_of_warm_up_epochs=None*, *log_directory=None*, ***kwargs*)

Gaussian-mixture variational auto-encoder class.

Parameters

- **feature_size** (*int*) – The number of features/genes in the data set to model.
- **latent_size** (*int*) – The number of dimensions to use for the latent space.
- **hidden_sizes** (*list(int)*) – A list of the number of units in each hidden layer of both the inference (encoder) and the generative (decoder) networks. The number of layers in each network is thus the length of this list. For the inference network, the order of the hidden layers is the same as for the list, while for the generative network, it is the reverse.
- **reconstruction_distribution** (*str, optional*) – The name of the reconstruction distribution (or likelihood function; see *Training a model*)
- **number_of_reconstruction_classes** (*int, optional*) – The number of counts to model directly, starting from zero (see *Training a model*).
- **latent_distribution** (*str, optional*) – The name of the latent prior distribution: `"gaussian_mixture"` or `"full_covariance_gaussian_mixture"` (see *Training a model*).
- **prior_probabilities_method** (*str, optional*) – Method for how to set the mixture coefficients for the latent prior distribution: `"uniform"` distribution, `"custom"` (provide probabilities to `prior_probabilities`), or `"learn"` during training.
- **prior_probabilities** (*1-d array-like, optional*) – Prior probabilities required when `prior_probabilities_method` is `"custom"`.
- **number_of_latent_clusters** (*int, optional*) – The number of latent clusters, which is also the number of components in the Gaussian-mixture model.
- **minibatch_normalisation** (*bool, optional*) – If `True`, normalise each random minibatch of data when training or evaluating the model.
- **batch_correction** (*bool, optional*) – If `True`, and if batches are present in data set to model, perform batch correction.

- **number_of_batches** (*int, optional*) – The number of batches in the data set to model. Required, if `batch_correction` is `True`.
- **number_of_warm_up_epochs** (*int, optional*) – The number of epochs during the start of training with a linear weight on the KL divergence. This weight is gradually increased linearly from 0 to 1 for this number of epochs.
- **log_directory** (*str, optional*) – Directory where model is saved.

feature_size
The number of features/genes which can be modelled.

latent_size
The number of dimensions of the latent space.

hidden_sizes
A list of the number of units in each hidden layer of both the inference (encoder) and the generative (decoder) networks. The number of layers in each network is thus the length of this list. For the inference network, the order of the hidden layers is the same as for the list, while for the generative network, it is the reverse.

reconstruction_distribution
An instance of the reconstruction distribution (or likelihood function) class used by the model.

number_of_reconstruction_classes
The number of counts modelled directly, starting from zero.

latent_distribution
An instance of the latent prior distribution class used by the model.

prior_probabilities_method
Method for how the mixture coefficients for the latent prior distribution are set: `"uniform"` distribution, `"custom"` (given by `prior_probabilities`), or `"learn"` during training.

prior_probabilities
Prior probabilities when `prior_probabilities_method` is `"custom"`.

minibatch_normalisation
If `True`, normalise each random minibatch of data when training or evaluating the model.

batch_correction
If `True`, and if batches are present in data set to model, perform batch correction.

number_of_batches
The number of batches in the data set to model, when `batch_correction` is `True`.

number_of_warm_up_epochs
The number of epochs during the start of training with a linear weight on the KL divergence. This weight is gradually increased linearly from 0 to 1 for this number of epochs.

property name
Short name for model used in filenames.

property description
Description of model.

property parameters
Trainable parameters in the model.

property number_of_latent_clusters
The number of latent clusters used in the model.

train(*training_set*, *validation_set=None*, *number_of_epochs=None*, *minibatch_size=None*, *learning_rate=None*, *run_id=None*, *new_run=False*, *reset_training=False*, ***kwargs*)
Train model.

Parameters

- **training_set** (DataSet) – Data set used to train model.
- **validation_set** (DataSet, *optional*) – Data set used to validate model during training, if given.
- **number_of_epochs** (*int*, *optional*) – The number of epochs to train the model.
- **minibatch_size** (*int*, *optional*) – The size of the random minibatches used at each step of training.
- **learning_rate** (*float*, *optional*) – The learning rate used at each step of training.
- **run_id** (*str*, *optional*) – ID used to identify a certain run of the model.
- **new_run** (*bool*, *optional*) – If `True`, train a model anew as a separate run with an automatically generated ID.
- **reset_training** (*bool*, *optional*) – If `True`, reset model by removing saved parameters for the model.

sample(*sample_size=None*, *minibatch_size=None*, *run_id=None*, *use_early_stopping_model=False*, *use_best_model=False*)
Sample from trained model.

Parameters

- **sample_size** (*int*, *optional*) – The number of samples to draw from the model.
- **minibatch_size** (*int*, *optional*) – The size of the random minibatches used at each step of training.
- **run_id** (*str*, *optional*) – ID used to identify a certain run of the model.
- **use_early_stopping_model** (*bool*, *optional*) – If `True`, use model parameters, when early stopping triggered during training. Defaults to `False`.
- **use_best_model** (*bool*, *optional*) – If `True`, use model parameters, which resulted in the best performance on validation set during training. Defaults to `False`.

Returns A data set of generated examples/cells as well as a dictionary of data sets of samples for the two latent variables.

evaluate(*evaluation_set*, *minibatch_size=None*, *run_id=None*, *use_early_stopping_model=False*, *use_best_model=False*, ***kwargs*)
Evaluate trained model

Parameters

- **evaluation_set** (DataSet) – Data set used to evaluate model.
- **minibatch_size** (*int*, *optional*) – The size of the random minibatches used at each step of training.
- **run_id** (*str*, *optional*) – ID used to identify a certain run of the model.
- **use_early_stopping_model** (*bool*, *optional*) – If `True`, use model parameters, when early stopping triggered during training. Defaults to `False`.

- **use_best_model** (*bool, optional*) – If `True`, use model parameters, which resulted in the best performance on validation set during training. Defaults to `False`.

Returns A data set of reconstructed examples/cells as well as a dictionary of data sets of the two latent variables.

1.3.3 Analyses module

scvae.analyses.**analyse_data**(*data_sets*, *decomposition_methods=None*, *highlight_feature_indices=None*, *analyses_directory=None*, ***kwargs*)

Analyse data set and save results and plots.

Parameters

- **data_sets** (*list (DataSet)*) – List of data sets to analyse.
- **decomposition_methods** (*str or list (str)*) – Method(s) used to decompose data set values: `"PCA"`, `"SVD"`, `"ICA"`, and/or `"t-SNE"`.
- **highlight_feature_indices** (*int or list (int)*) – Index or indices to highlight in decompositions.
- **analyses_directory** (*str, optional*) – Directory where to save analyses.

scvae.analyses.**analyse_model**(*model*, *run_id=None*, *analyses_directory=None*, ***kwargs*)

Analyse trained model and save results and plots.

Parameters

- **model** (*(GaussianMixture)VariationalAutoencoder*) – Model to analyse.
- **run_id** (*str, optional*) – ID used to identify a certain run of `model`.
- **analyses_directory** (*str, optional*) – Directory where to save analyses.

scvae.analyses.**analyse_intermediate_results**(*epoch*, *learning_curves=None*, *epoch_start=None*, *model_type=None*, *latent_values=None*, *data_set=None*, *centroids=None*, *model_name=None*, *run_id=None*, *analyses_directory=None*)

Analyse reconstructions and latent values.

Reconstructions and latent values from evaluating a model on a data set are analysed, and results and plots are saved.

Parameters

- **evaluation_set** (DataSet) – Data set used to evaluate `model`.
- **reconstructed_evaluation_set** (DataSet) – Reconstructed data set from evaluating `model` on `evaluation_set`.
- **latent_evaluation_sets** (*dict (str, DataSet)*) – Dictionary of data sets of the two latent variables.
- **model** (*(GaussianMixture)VariationalAutoencoder*) – Model evaluated on `evaluation_set`.
- **run_id** (*str, optional*) – ID used to identify a certain run of `model`.
- **sample_reconstruction_set** (DataSet) – Reconstruction data set from sampling `model`.

- **decomposition_methods** (*str or list(str)*) – Method(s) used to decompose data set values: `"PCA"`, `"SVD"`, `"ICA"`, and/or `"t-SNE"`.
- **highlight_feature_indices** (*int or list(int)*) – Index or indices to highlight in decompositions.
- **early_stopping** (*bool, optional*) – If `True`, use parameters for `model`, when early stopping triggered during training. Defaults to `False`.
- **best_model** (*bool, optional*) – If `True`, use parameters for `model`, which resulted in the best performance on validation set during training. Defaults to `False`.
- **analyses_directory** (*str, optional*) – Directory where to save analyses.

1.3.4 Argument defaults

Below are listed the defaults for some optional arguments:

```
{
        "data": {
                "format": "infer",
                "directory": "data",
                "map_features": false,
                "feature_selection": [],
                "example_filter": [],
                "preprocessing_methods": [],
                "noisy_preprocessing_methods": [],
                "split_data_set": false,
                "splitting_method": "default",
                "splitting_fraction": 0.9
        },
        "analyses": {
                "directory": "analyses",
                "decomposition_method": "PCA",
                "decomposition_dimensionality": 2,
                "highlight_feature_indices": [],
                "included_analyses": "standard",
                "analysis_level": "normal",
                "export_options": []
        },
        "models": {
                "directory": "models",
                "type": "VAE",
                "latent_size": 2,
                "hidden_sizes": [100],
                "number_of_samples": {
                    "training": 1,
                    "evaluation": 1
                },
                "latent_distribution": {
                        "VAE": "gaussian",
                        "GMVAE": "gaussian mixture"
                },
                "number_of_classes": 1,
                "parameterise_latent_posterior": false,
                "inference_architecture": "MLP",
                "generative_architecture": "MLP",
                "reconstruction_distribution": "poisson",
```

(continues on next page)

(continued from previous page)

```
                "number_of_reconstruction_classes": 0,
                "prior_probabilities_method": "uniform",
                "number_of_warm_up_epochs": 0,
                "kl_weight": 1,
                "proportion_of_free_nats_for_y_kl_divergence": 0.0,
                "minibatch_normalisation": true,
                "batch_correction": false,
                "dropout_keep_probabilities": [],
                "count_sum": false,
                "number_of_epochs": 200,
                "minibatch_size": 100,
                "learning_rate": 1e-4,
                "sample_size": 0,
                "run_id": "",
                "new_run": false,
                "reset_training": false
        },
        "evaluation": {
                "data_set_kind": "test",
                "prediction_training_set_kind": "training",
                "prediction_method": "",
                "model_versions": "all"
        },
        "cross_analysis": {
                "log_summary": false
        }
}
```

1.4 References

Christopher Heje Grønbech, Maximillian Fornitz Vording, Pascal N. Timshel, Casper Kaae Sønderby, Tune H. Pers, and Ole Winther (2020). "scVAE: Variational auto-encoders for single-cell gene expression data". *Bioinformatics*, btaa293. Diederik P. Kingma and Max Welling (2014). "Auto-encoding variational Bayes". *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Diederik P. Kingma and Jimmy Ba (2015). "Adam: A method for stochastic optimization". *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models". In: Xing, E.P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, PMLR, pp. 1278–1286.

PYTHON MODULE INDEX

S

INDEX